Ascribing Artificial Intelligence to (Simpler) Machines
or
When AI Meets the Real World

Robert E. Filman

IntelliCorp, Inc.
1975 El Camino Real West
Mountain View, California 94040

Introduction

John McCarthy's contributions to Computer Science are legion. He is not only the father of Artificial Intelligence, but also (among other things) the originator of symbolic and functional programming (Lisp), the first to define and explore the mathematical semantics of programs, an innovator in the development of conventional programming languages (Algol), and the creator of the first timesharing operating system. McCarthy's most important child, AI, is now no longer purely a scientific curiosity. AI (or at least something called AI) has moved out into the (commercial) world and is "making a living," so to speak. One now hears people freely ascribing "Artificial Intelligence" to computer programs, often with little regard to the underlying intelligence of the system. In this chapter, I discuss (my understanding of) McCarthy's perspective on systems and AI and how this point of view has shaped the commercial AI market. I focus on what we have done at IntelliCorp, in product and research, both inside and outside the McCarthy tradition. I close with a few comments on where I believe the profitable confluence of theoretical and applied AI lies in the (near) future.

SAIL and the FOL Project

I had the excellent fortune of working for John at the Stanford Artificial Intelligence Laboratory (SAIL) in the mid-to late '70s, roughly (most of) the period 1974-79. I confess I found SAIL to be a magical place. The lab was located on the fringe of the Stanford

campus, surrounded by (usually) golden hills and guarded by a diamond robot-vehicle caution. The building was a labyrinth of halls and offices, notable both for its unusual architecture (primarily wood construction on a steel frame, 1/3 circle, with boardwalk and picture windows, centered on a volleyball court—sort of industrial Eichler) and for the extraordinary care the university gave to maintaining the physical establishment. SAIL was far removed from the tempo of ordinary life—the only hints of the passage of time were the 4 p.m. calls for volleyball and the sun rising over the hillside in the morning. The main campus was another continent, ARPA an unquestioningly generous but remote benefactor, and commerce and industry a different universe.

Of course, for computer-techies, much of the magic of SAIL lay in the computational facilities. Prior to working at SAIL, my experience with computers had been mediated through punch cards and teletypes. The AI lab was a world of interactive display and visual editing; hardcopy graphic output (XGP); robot eyes, arms, and carts; electronic mail; spacewar, paranoia and adventure; computer-vended Chinese food; and television or radio on your office terminal, all while your program spun round the coffee tables in the machine room. We had sixty people interactively display editing and running programs on a machine slower and smaller than a single current (1991) workstation. We had XGP output at a time when computers printed on line printers and the only way to get fonts or graphics was to be a typesetter. We got the news, sorted and filtered, before the newspaper editors. Now, of course, anyone can have more pure computational power for a few thousand dollars; the exceptional quality of the SAIL environment may be lost on the younger generation.

I believe a critical theme of these facilities—one that was necessary for their existence and evolution—was interaction. At SAIL, we explored the "world" described or mediated by a computer system (be it a world of programs, text, or robots). The problems we worked on were too difficult to understand *a priori*.

SAIL fostered a distinctly unconventional culture. What mattered was the quality of your science (or the cleverness of your hacks), not, say, your appearance, charm, or the financial consequences of what you were doing. SAIL, to a large extent because it ignored the strictures of commerce, created great science (and legendary hacks). I have seen enough of the rest of the world to understand that such creative environments come about only as a reflection of the people in charge of them. McCarthy understood what mattered and what did not. He fostered an environment of independence, play, and research that was far more productive than one of schedules, profits, and directed work would have been.

FOL

Most of my work at SAIL was on the FOL project. FOL was a "proof checker for first-order logic" [Weyhrauch80]. Using FOL involved two steps: describing the representation of some domain in first-order logic and sequentially stating the steps of a proof of a desired conclusion. FOL would check to make sure that these steps were valid. FOL also did additional bookkeeping, such as keeping track of the assumptions underlying any particular conclusion.

FOL dealt with a many-sorted first-order logic. One could define a language: predicates of fixed arity, constants, variable symbols, and functions. Single-argument predicates defined types (sorts). Constants, variables, and the arguments and results of predicates and functions could be "restricted" to particular sorts. There were facilities

for developing sort hierarchies and for overloading predicates and functions. Using the usual connectives, quantifiers, and equality, one built sentences in the language. Some of these would be declared axioms.

FOL began as a natural-deduction-style proof checker. Thus, the most primitive operations involved the introduction and elimination of connectives and quantifiers—for example, the instantiation of a universally quantified axiom to particular individuals. FOL was extended by a tautology checker, a tautology checker for equality, a decision procedure for the monadic predicate calculus, a syntactic simplifier (for syntactic substitutions), semantic attachment (the ability to describe a correspondence between the symbols in the linguistic space and a computational model, deriving conclusions based on computations in the model), and, perhaps most interestingly, a reflection mechanism for reifying the structures in an FOL system and reasoning about them metatheoretically. Pushed to its limit, this last mechanism resulted in context structures that could be reasoned about at increasingly higher meta-levels. Using FOL, we developed representations and proofs of problems in domains such as program semantics, pure mathematics, digital circuits, puzzles, and chess.

FOL was thus very much in the tradition of—one might fairly say an early instantiation of—McCarthy's Advice Taker [McCarthy58]. It was a system to which one described a domain and asserted consequences in that domain. Like the Advice Taker, FOL would check these assertions for validity. Over time, the complexity of the inferential steps it could confirm increased greatly. However, its heuristic powers were relatively limited—it relied on the human to direct the flow of the conversation.

The FOL system illustrated several key themes in McCarthy's work:

- The idea that the world can be represented symbolically for a machine and reasoned about by that machine.
- The division of the AI problem into two parts: epistemological (representation) and heuristic (search).
- The importance of formal (mathematical) representation, particularly the use of logic for representation.
- The use of *reification*—the conceptual turning of abstract structures (such as situations) into concrete instances that can be reasoned about. The situation calculus, where the state of the universe is compressed into a single constant, is an example of such reification [McCarthy69].
- The development of abstract and general mechanisms (proof checkers, logic, circumscription, common sense) in contrast with domain-specific tools and languages.
- The relatively complete analysis of artificial domains (programming languages, puzzles, chess) instead of a (necessarily) incomplete analysis of natural situations.
- The importance of environments for interactive exploration. This is illustrated not so much by an explicit campaign position of McCarthy's as by the systems whose creation he led, such as the AI lab, Lisp, and timesharing.

First Commercializations

In the early '80s, AI, with considerable hype, began its transition out of the AI labs and into the marketplace. Initially, the dominant marketing buzzword for AI programs was "expert systems." As time wore on and the difficulty of true expertise became apparent, much of the market retreated to other slogans, such as the one I will use, "knowledge-based systems" (KBS). The primary vehicles for the commercialization of AI became tools for building KBS. These tools embody the computer science dictum that if you *really* know how to do something, you build a program that implements that knowledge. "All" that we knew about AI was the general utility of certain structures and algorithms. These tools package those structures and algorithms.

KBS tools can be understood as taking McCarthy's use of general-purpose first-order logic for knowledge representation and cutting out the computationally infeasible parts. What remains has considerably less expressive power but can be implemented efficiently. More specifically:

- The individual constants of logic become *objects*, implemented by some specific data structure. While puzzle domains often turn on establishing the identity of two individuals, these tools assume that different objects represent different individuals. Although establishing the class of an object has a place in the use of KBS tools, establishing the identity of two objects does not. Certain primitive individuals (such as numbers) are second-class entities.

- First-order logic supports many arity predicates. These tools reduce the world to monadic and binary relations.
  - Monadic relations on objects become *classes*. *Subclass* and *element of* (instance) are primitives.
  - Binary relations become *slots* on objects. The computational effect of slots is to relations by their first argument. It becomes easy to find the **color** of **Clyde** or, for that matter, every known property of **Clyde**, more difficult to find the **color** of every **elephant**, and very inefficient to find everything that has some relationship with **pink**. Some aspects of higher-arity predicates can be simulated by reifying a true instance of the predicate as an object and using its slots for the arguments.

- The system provides a special inference mechanism, *inheritance*. Inheritance automates the conclusion of $P(e,x)$ from $e \in C \wedge P(C,x)$ (and $P(S,x)$ from $S \subseteq C \wedge P(C,x)$) for arbitrary relations (slots) P. More interestingly, such systems often implement a variety of truth-maintained, nonmonotonic logic, where the above implication holds unless there is a reason (like an externally specified value for P on e) for it not to hold.

- The expressibility of the full logic for axioms is reduced to universally quantified implications over an unquantified logic. Often the class of legal axioms is further restricted, for example, by eliminating negation, by treating negation as a thnot ("can't find"), or by allowing only axioms in Horn clause form. In the jargon of knowledge-based system tools, these become *rules*. A primary inference technique is to successively instantiate rules to particular individuals. This can be done either by computing the closure over a set of rules produced by a set of facts (*forward chaining*) or by seeking a constructive proof of the satisfaction of some particular clause (*backward chaining*).

A dominant theme of this decomposition is taking abstract quantified statements and reducing them to statements about concrete instances early in the inference process. There is rarely an operation that produces another abstraction. Similarly, those abstract statements that are allowed are restricted to a semantically straightforward logic.

Add in a graphic, interactive user interface and a couple of programmatic interventions—the ability to invoke specific algorithms for the values of slots (access-oriented programming; essentially, our old friend semantic attachment) and the ability to overload programs by their first argument (object-oriented programming) and you get a first-generation knowledge-based system development tool, such as the first or second version of the KEE™ system[1] [Kehler84, Fikes85].

Next-Generation Systems

I arrived at IntelliCorp at about the same time as we (the IntelliCorp research group) began (under DARPA funding) to extend these ideas. Initially, our work focused on incorporating two additional utilities to the KEE™ 2 base:

- A truth-maintenance system, based on de Kleer's Assumption-based Truth Maintenance System (ATMS) [Kleer86].
- A context mechanism, implemented using ATMS mechanisms, which drew its semantics from classical AI planning problems.

These utilities became key elements of the third version of KEE™.

KEE™ 3

An AI system must have domain of "possible facts" that it "can believe." Let us call such sentences *propositions*. A typical AI system starts with a set of propositions, its *axioms* or background truths. It builds justifications for propositions by (1) making *assumptions* and (2) drawing inferences based on these assumptions. Each such construction is a proof step. When belief in an assumption is no longer appropriate, it is necessary to withdraw belief in the conclusions based on that assumption. The purpose of a truth maintenance system (TMS) is to automate this last step—that is, a TMS remembers the reasons for belief of each conclusion and deletes (or perhaps inhibits the visibility of) those conclusions whose assumptions are no longer valid. Thus, an important component of any TMS is that it records the dependencies (i.e., proofs) of each conclusion.

There are two popular styles of TMSs—de Kleer's ATMS and the Doyle style [Doyle79, London78]. A Doyle-style TMS incorporates the idea that particular assumptions can be *in* (currently believed) or *out* (not currently believed) at any time. A particular derivation would be valid, for example, if assumptions *X* and *Y* were *in* but *Z out*. In-

[1] My examples are drawn from KEE™ as I am most familiar with that system. Of course, several other systems share similar properties. KEE™ 2 ran on dedicated Lisp machines. Its users praised the quality of interaction, the ease of building graphic applications, and the uniformity of representation. KEE™ 2 was used primarily by research laboratories to build demonstrations of systems of an intellectual complexity greater than prior tools had allowed. Many called these programs examples of Artificial Intelligence; the availability of such tools was (circularly) a primary reason why building AI applications became routine events.

ness and out-ness enable both modeling a "current world" (worlds being assignments of in and out to assumptions) and basing beliefs on the out-ness of facts. A Doyle-style TMS thus implements a particular nonmonotonic logic. Computationally, a Doyle-style TMS is efficient at finding if a proposition is in the current set of beliefs, but can be inefficient at changing to a new set of beliefs.

De Kleer's ATMS, rather than pushing facts *in* and *out*, keeps every proof of each conclusion. Queries to the knowledge base are given with respect to a particular set of assumptions; only those conclusions that follow from the given assumptions are visible to such a query. The particular cleverness of the ATMS algorithm is that (1) it caches (usually as a bit vector) the sets of assumptions that lead to proofs of each conclusion and (2) it discards inconsistent and subsumed assumption sets. An ATMS is efficient at switching and comparing belief sets. It has the interesting property that a conclusion can be used not only in the context in which it was derived, but also in any other context that shares the assumptions underlying that conclusion. The ATMS has the weaknesses of requiring some search to retrieve the facts associated with a particular assumption set and a tendency to be storage-intensive. Note the similarities of FOL and the ATMS: like the ATMS, FOL keeps track of proofs and dependencies. Additionally (and interestingly from the point of view of truth maintenance), both FOL and the ATMS are monotonic.

Using the ATMS as a foundation, we built a context mechanism, much in the spirit of the contexts of QA4 [Rulifson72] and Conniver [McDermott73]. We call each context a *world*. Worlds can have multiple parents and children; the primary limitation is that the parents of a world be created before that world. (This ensures that the world graph is acyclic.) Creating a new world creates a new ATMS assumption (the *world assumption*) that asserts (effectively) that the world exists (or, alternatively, the decision to create that world was made). Associated with the world is the *world environment*—the union of the world assumption of the world and its ancestors. This is implemented by creating an ATMS proposition (effectively, "this world exists") and providing the world environment as its only justification. Asserting a proposition in a world involves (1) creating an assumption to represent "that fact is believed in this world" (the non-deletion assumption) and (2) justifying the proposition by the union of the non-deletion assumption with the world environment. This allows the fact to be later deleted from the world by making the non-deletion assumption contradictory. Testing the context-relative belief in a proposition is straightforward—if the world environment, coupled with all consistent non-deletion assumptions, is a superset of the assumption set in any proof of that proposition, that proposition is believed in that world. Since worlds are many-parented, a mechanism is required to resolve ambiguities between parent additions and removals of belief in individual propositions. The mechanism chosen in KEE™ is to treat worlds as the situations of a planning process and to include the fact if some linearization of the creation order of the parent worlds results in the proposition holding in the child. The details of this implementation are discussed in [Morris86]; its use in [Filman88].

How does the ATMS/worlds system correspond to the kinds of things one can say in a system such as KEE™? We first note that KEE™ makes the distinction between *own* slots—slots that are properties of the object in question, and *member* slots—slots that are properties of the members of a class. KEE™ provides several inheritance roles (the way of combining a slot's values with its parents' values). Thus, if class *C* has a *member* slot *S* with values $v_1, \ldots, v_n$, its descendant instance, *e*, will have an *own* slot *S*. The values of *S* in *e* are a function of the inheritance role of the slot, the v_i, and *e*'s local values on *S*. (If *e* has multiple ancestors with *S*, the inheritance role combines their values.)

For the first version of our system, we chose to restrict the propositions that could be used in ATMS justifications to non-inheritable values of slots—own slot values—and to an auxiliary database of arbitrary forms apart from the frame system (much like a Prolog database), the *unstructured facts*. We excluded, for lack of time and technique, truth maintenance of other kinds of information. Propositions supported by the ATMS were encoded on the slot data structure. The primary mechanism (at least in the minds of most users) for building justifications is a justification-building rule (in KEE™ terms, a *deduction rule*). These rules have the property of being side-effect-free. It is also possible to programmatically create justifications.

Pleased with the power of these mechanisms, IntelliCorp rushed to commercialize them. The ATMS and worlds mechanism became a major component of the third version of KEE™. I found it striking as to how quickly these novel AI algorithms had become generally and commercially available. Alas, in practice, almost all KEE™ users ignore these features.

OPUS

A lack of immediate commercial acceptance did not faze our group. After all, we had an academic bent and a charter to explore the *next* generation of knowledge-based tools. Our first step was an attempt to make the ATMS mechanisms pervasive throughout the frame system. KEE™ 3 could associate propositions with own slot values and unstructured facts. We wanted to make the other facts of the frame system—member slot values, inheritance links, facet values, and the existence of slots and objects—also be subject to truth maintenance. We called this system OPUS [Fikes87].

Our initial tactic was to minimize the amount of modification required by increasing the system's structural uniformity. We accomplished this by (1) transforming inheritance links to slots and (2) creating recursive objects—that is, the slots of objects became implemented themselves as objects, facets being interpreted as slots on the slot objects. The result of this transformation was to make every fact about the frame system that concerned the ATMS the value of some slot. (In practice, not every slot value needs to have an ATMS justification. Most facts remain in the *background*, unencumbered by the ATMS mechanism.)

The last two issues (object and slot existence) are not intellectually difficult—they depend only on creating propositions that state that the object or slot exists and justifying these propositions like any others. However, this requires modifying almost every function in the system to check these assertions. We viewed the volume of effort to make these changes as outweighing the presumed benefit—in our system, object and slot existence remained universal.

The more difficult issues arose in unifying the ATMS, a monotonic system, with inheritance, a nonmonotonic one. Broadly, a value in a member slot can mean one of three things: (1) That this value is true of all children of the object—that is, that this is a *necessary value* for the slot. Necessary values cause no problems, as they have a monotonic semantics. (2) That this is a *default value* for the slot, to be used unless something blocks use of the default (or, in more McCarthyesque terms, if the situation is somehow *abnormal*). This is a nonmonotonic inference. (3) That this value is to be used as input to an operational inheritance role—that is, one whose semantics is defined by a program. While most of KEE™'s inheritance roles are operational, the programmatic nature of such roles makes them ill-suited for truth maintenance.

The challenge thus became to express default information in a way compatible with an inference process and to extend the ATMS to deal efficiently with this nonmonotonic mechanism. As the basis of our algorithm, we used a variant of an exception mechanism of Doyle and Touretzky [Touretzky86]. This required splitting member slot values into necessary and default values. Necessary values inherit unmodified to all children. Default values require some way of excluding the default. We realized this by breaking the defaults into *default values* and *exceptions*. Conceptually, defaults inherit through the hierarchy unless blocked by an exception. Exceptions are pairs, a class and a value, implying that the specified value is not to be inherited from that class. Both the class and the value can be universal, providing the ability to defeat some or all defaults from some or all ancestor classes [Nado86].

For each type of value or exception on a slot, the inheritance algorithm stores two kinds of values: *primitive* values that have been explicitly set by user action and *derived* values that have been deduced as a combination of the primitive values, inherited values, and universally ATMS-justified values. This requires splitting all ATMS propositions into three classes: *in*, those that are believed in all contexts; *out*, those that are believed in no context; and *world-dependent*, those that are believed in some (but not all) contexts. The inheritance algorithm takes these labelings into account, propagating only inherited values that are globally valid. ATMS demons associated with label updating inform the inheritance algorithm of when the *in* status of a slot value or the *world-dependent* or *out* status of an exception changes. Correspondingly, the inheritance algorithm supplies or retracts justifications to the ATMS when a primitive value is asserted or removed from a slot, or a derived value is determined or deleted.

To support the default rules of the inheritance mechanism, the ATMS must manipulate nonmonotonic justifications—that is, justification expressions involving *out* justifiers. We want nonmonotonic justifications to hold in worlds whose *in* justifiers are satisfied and whose *out* justifiers are not. The basic ATMS does not provide nonmonotonic justifications; we must simulate them. We do this by creating a proposition *out*(*F*), for each *out* justifier *F*. We justify it so that *out*(*F*) is true in those worlds where *F* is not true. This is accomplished by (1) phrasing justifications in terms of out justifiers and (2) developing a justification (perhaps involving generated, intermediate propositions and justifications) expressing the total state of knowledge about the nonmonotonic belief in a proposition. This justification, in a way reminiscent of non-deletion assumptions, includes a *closed* justifier that can be falsified if additional justification for the belief or non-belief of a proposition is asserted.

Like a Doyle-style TMS, the mechanism is complicated by circular justifications (e.g., where $A \supset B$ and $B \supset A$). This difficulty is resolved by including only well-founded support for propositions. We take advantage of the ATMS labeling algorithm to determine well-foundedness. Correspondingly, even cycles (e.g., $out(A) \supset B$ and $out(B) \supset A$) and odd cycles (e.g., $out(A) \supset A$) of nonmonotonic justifications require special mechanisms, the first handled by treating $out(out(X)) \equiv X$, and the second by deeming them inconsistent [Fikes87].

The OPUS system can be understood as an attempt to realize in a KBS tool some of the more sophisticated formal AI ideas about nonmonotonic reasoning. Two things are noteworthy about the resulting system: (1) Given the size and speed of current computers, the storage and processing requirements of these algorithms render them unusable for all but the simplest problems, and (2) these mechanisms are far more sophisticated than what our customers' applications need.

Other Work

It is worth mentioning that under the next-generation systems charter, we explored several other topics involving tools for knowledge-based systems, including composite objects; multi-agent problem solving; concurrency mechanisms (on both conventional networks and connection machines); tools for object-oriented static program analysis; alternative implementations for frames; and symbolic languages that combine sequential action, pattern-matching, and chaining [Filman87, Mishelevich88, Filman89]. Only the last two of these have had, to date, significant product impact. In general, what seems to matter to the users of the current generation of KBS tools is primarily (1) integration with existing systems, (2) quality of interface, and (3) efficiency. It is enough to have only a little "artificial intelligence."

Discussion

I noticed the following in this morning's (February 28, 1991) business section of the *San Jose Mercury*. James Mitchell, the business editor, was discussing how a particular manufacturer achieves a high level of mainframe reliability. Tossed in three-quarters of the way down the column, I read:

> Using artificial intelligence routines, Hitachi Data's customer support center analyzes the raw data, automatically determines the necessary action and alerts the customer service representative.

I cite this quote to illustrate that (some) people are now willing to casually ascribe (some) degree of artificial intelligence to machines. Just as McCarthy observed the anthropomorphization of a thermostat [McCarthy79], it has become common to credit systems that have some symbolic reasoning facilities with Artificial Intelligence. Are they intelligent? Only in the most restricted ways. Nevertheless, using tools like KEE™, people develop systems that demonstrate useful "intelligence" in a variety of areas, e.g., scheduling complex processes, configuring intricate systems, diagnosing difficult irregularities, and, most unexpectedly, serving simply as Advice Takers, checking decisions for constraint satisfaction without actually suggesting particular courses of action. What matters to the developers of these systems are a useful set of representational primitives, good inference mechanisms, and (surprisingly to many coming from a formal background) high-quality interaction. What matters only infrequently are representation mechanisms able to express fine logical nuances.

The moral of this tale is that it takes considerably less technology than a complete artificial intelligence to do useful and interesting things in the world. This level of success has been achieved by taking John McCarthy's vision of general-purpose, symbolic, logic-based reasoning systems and sharply reducing the generality and expressiveness until what remains is computationally tractable though somewhat boring.

The current generation of KBS tools has reached a plateau, where the performance and expressiveness is adequate for a simple degree of "artificial intelligence." (That is, I hope the current state of affairs is a plateau). Current systems are a distillation of the research from AI labs in the '60s and '70s. I would like to close by suggesting a few places where I think an appropriate confluence of theoretical ideas and applications will take us to the next generation of KBS tools. Coming from a McCarthyesque tradition, my top-

ics are concerned with general-purpose mechanisms, not techniques most appropriate for a particular class of applications.

Composite objects The object-centered approaches to KBS tools glibly divide the world into discrete instances, with the assumptions that (1) all individuals of interest can be explicitly mentioned and (2) inheritance is a sufficient mechanism for describing all default information about individuals. Effectively, systems like KEE™ reify only classes and elements of those classes. However, many real problems call for a shifting of boundaries—a set of objects may be referenced collectively in one part of the problem, then broken down into anonymous or distinguished components, some of which may then be reassembled into different composite objects. (This problem comes up frequently, for example, in configuration applications, where subcomponents are numerous and the particular identity of subcomponents is not important until you have something specific to say about them.) Importantly, some but not all properties carry over from the composite objects to their subparts, while other properties of the composites are assembled from their elements. Note that it is straightforward to express such concepts in a first-order logic with functions and numeric constraints on the number of values of a multi-function. The challenge for tool builders is to develop efficient algorithms and representation mechanisms for implementing composite objects in KBS tools. (We made an initial stab at such a mechanism in some of our DARPA work [Mishelevich88]; clearly, much more needed to be done.) I think it is likely that such algorithms will need to mutate conventional KBS tools to include notions of equality and lazy object realization.=1

Generalized annotation In this chapter, I have examined in detail some mechanisms for associating proofs with the facts of a knowledge base. Of course, in making this modification, we were required to recreate the system's inference mechanisms: inheritance and rule chaining. I have not mentioned another popular variety of KBS mechanism, the association of certainty factors or probabilities with knowledge-base facts [Shortliffe76, Pearl88]. Of course, a certainty factor system requires a rule mechanism of its own. Both of these are examples of a more general phenomena, the *annotation* of knowledge-base information. Other examples of profitable points of annotation include temporal information (when is this true?), source information (who said it?), and heuristic information (when is this information appropriate to use?). Annotation is a specialization of an old FOL friend of the '70s, metatheoretic reasoning, where one can annotate not merely ground-level facts but arbitrary axioms [Weyhrauch80]. The challenge is to develop efficient systems and inference mechanisms for which such annotation is declarative—where the user can described the desired types of annotation and their combination and control of the inference mechanism, instead of relying only on the particular annotation mechanisms the system developers implemented.

Symbolic optimization In many ways, the major AI problem with respect to the use of KBS tools is not one of representation. Despite their ingenuousness in comparison with the full first-order logic, objects, slots, and inheritance are a useful set of mechanisms *with a clear methodology for their use.* However, what continually amazes me is how little support KBS tools provide for the heuristic part of the AI problem. We seem to have stopped at blind backtracking. The only additional AI search

mechanisms in KBS tools are rule systems, which are fundamentally non-deterministic programming. Non-deterministic programming is difficult for skilled programmers, no less the users of KBS tools. I believe we are seeing some progress in this respect and have an opportunity for more. Currently, there is much work on the development of constraint-based systems. Such systems take problems specified as a set of variables, a set of possible values for those variables (drawn from discrete or numerically continuous sets), and a set of constraints (invalid variable-value combinations) and discover assignments to the variables that do not violate the constraints. But in real problems, most constraints are not "hard"—rather, many solutions may be legal, but some are *better* than others. I believe the next step in this technology ought to be the development of symbolic optimizers—systems that can take a problem, a set of constraints and a partially defined utility function (e.g., "Higher *x* is better than lower") and present to the user a *frontier* of possible solutions. Such a symbolic optimizer could be driven by some form of hill-climbing or simulated annealing, perhaps compiling well-defined subproblems into the algorithms of Operations Research.

Acknowledgments

I have been quite liberal in this chapter in discussing the work of groups in which I have been only one of many participants. Particularly in a volume whose purpose is somewhat historical, it is important to acknowledge those other individuals. My coworkers on the FOL project included (of course) John McCarthy, Richard Weyhrauch, Dan Blom, Juan Bulnes, Bill Glassmire, Chris Goad, Andrew Robinson, Carolyn Talcott, Arthur Thomas, and Todd Wagner. At IntelliCorp, our research group has included Conrad Bock, Roy Feldman, Richard Fikes, Phil McBride, Paul Morris, Bob Nado, Anne Paulson, Josh Singer, Richard Treitel, and Martin Yonke. My thanks to Bill Faught, John Kunz and Paul Morris for comments on drafts of this chapter.

Some of the work described here was supported by the Defense Advanced Research Project Agency under Contract F30602-85-C-0065. The views and conclusions reported here are those of the author and should not be construed as representing the official position or policy of DARPA, the U. S. Government, or IntelliCorp. KEE™ is a registered trademark of IntelliCorp, Inc.

References

[Dechter87] R. Dechter and J. Pearl, Network-Based Heuristics for Constraint Satisfaction Problems, *Artificial Intelligence* 34 (1987), 1-38.

[de Kleer86] J. de Kleer, An Assumption-Based Truth Maintenance System, *Artificial Intelligence* 28 (1986), 127-162.

[Doyle79] J. Doyle, A Truth Maintenance System, *Artificial Intelligence* 12 (1979), 231-272.

[Fikes85] R. Fikes and T. Kehler, The Role of Frame-Based Representation in Reasoning, *Comm. ACM* 28 (1985), 904-920.

[Fikes87] R. Fikes, R. Nado, R. Filman, P. McBride, P. Morris, A. Paulson, R. Treitel, and M. Yonke, OPUS: A New Generation Knowledge Engineering Environment: Phase 1 Final Report, IntelliCorp, Mountain View, California (1987).

[Filman87] R. E. Filman, Retrofitting Objects, in: Proc. ACM Conf. on Object Oriented Programming Systems, Languages, and Applications, Orlando, Florida (1987) 342-353.

[Filman88] R. E. Filman, Reasoning with Worlds and Truth Maintenance in a Knowledge-Based Programming Environment, *Comm. ACM* 31 (1988), 382-401.

[Filman89] R. E. Filman, C. Bock, R. Feldman, J. Singer, and R. Treitel, New Generation Knowledge System Development Tools: Final Report, IntelliCorp, Inc., Mountain View, California (1989).

[Kehler84] T. P. Kehler and G. Clemenson, An Application Development System for Expert Systems, *Syst. Softw.* 3 (1984), 212-224.

[London78] P. London,Dependency Networks as Representation for Modelling in General Problem Solvers, Technical Report 698, Department of Computer Science, University of Maryland (1978).

[McCarthy58] J. McCarthy, Programs with Common Sense, in: Mechanisation of Thought Processes, Proc. Symp. Nat. Phys. Lab., Her Majesty's Stationary Office, London (1958) Vol 1., 77-84. Reprinted in: M. Minsky, ed., *Semantic Information Processing*, The MIT Press, Cambridge, Massachusetts (1968) 403-418.

[McCarthy79] J. McCarthy, Ascribing Mental Qualities to Machines, in: M. Ringle, ed., *Philosophical Perspectives in Artificial Intelligence*, Humanities Press, Sussex (1979) 161-195.

[McCarthy69] J. McCarthy and P. Hayes, Some Philosophical Problems from the Standpoint of Artificial Intelligence, in: B. Meltzer and D. Michie, eds., *Machine Intelligence 4*, Edinburgh University Press, Edinburgh (1969) 463-502.

[McDermott73] D. V. McDermott and G. J. Sussman,The Conniver Reference Manual, Memo 259, MIT AI Laboratory, MIT, Cambridge, Massachusetts (1973).

[Mishelevich88] D. M. Mishelevich, R. E. Filman, C. Bock, P. Morris, A. Paulson, and R. Treitel,New Generation Knowledge System Development Tools: Phase 2 Interim Report, IntelliCorp, Inc., Mountain View, California (1988).

[Morris86] P. H. Morris and R. A. Nado, Representing Actions with an Assumption-Based Truth Maintenance System, in: Proc. AAAI-86, Philadelphia (1986) 13-17.

[Nado86] R. Nado and R. Fikes, Semantically Sound Inheritance for a Formally Defined Frame Language with Defaults, in: Proc. AAAI-87, Seattle (1987) 443-448.

[Pearl88] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Belief*, Morgan Kaufmann, San Mateo, California (1988).

[Rulifson72] J. F. Rulifson, A. Derksen and R. J. Waldinger, QA4: A Procedural Calculus for Intuitive Reasoning, Technical Note 73, Artificial Intelligence Center, Stanford Research Institute (1972).

[Shortliffe76] E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN*, American Elsevier, New York (1976).

[Touretzky86] D. S. Touretzky, *The Mathematics of Inheritance Systems*, Morgan Kaufmann, Los Altos, California (1986).

[Weyhrauch80] R. W. Weyhrauch, Prolegomena to a Theory of Mechanized Formal Reasoning, *Artificial Intelligence* 13 (1980), 133-170.